ATARI

Falcon030

AES 4.0
For MultiTOS

AES Documentation

3/27/92 - 11/19/92

==========

The following is a discussioy of new calls and new features in AES. Everything listed here is subject to change. If a feature is not clearly documented here, some more detail may be available in RELEASE.DOC. If it's not clear after reading both this file and RELEASE.DOC, that's a bug and should be reported.

The first part of this file is structured like the AES documentation, with "V4.0" introducing the section which describes new things for this version of the AES.

==========

3.4.4 APPL FIND

Purpose:

Finds the ap_id of another application in the system.

An application must know the ap_id before it can establish communications with another application.

V4.0

a. If the high word of ap_fpname is 0xFFFF, then the low word should contain the mint id of that application. The appl_find call will convert the mint id into the corresponding aes id.

b. If the high word of ap_fpname is 0xFFFE, then the low word should contain the aes id of that application. The appl_find call will convert the aes id into the corresponding mint id.

c. If ap_fpname is a zero pointer, the appl_find will return the aes id of the current process.

Parameters:

```
control[0] = 13
control[1] = 0
control[2] = 1
control[3] = 1
control[4] = 0

int_out[0] = ap_fid
addr_in[0] = ap_fpname
```

ap_fid = the ap_id of the application for which the current application is searching.

-1 - GEM AES could not find the application.

ap_fpname - address of a null-terminated string containing the filename of the application for which the current application is searching.

The string must be 8 characters long. If the filename has fewer than 8 characters, the programmer must fill out the rest of the string with blank spaces.

Sample call to C language binding:

```
ap_fid = appl_find( ap_fpname );
```

3.4.1. APPL INIT

Purpose:

Initalizes the application and establishes a number of internal GEM AES data structures prior to calls to other AES function calls.

V4.0

This must be the first call prior to all the other AES functions. Fail to do this will have unpredictable result!

The best way to detect if the AES is present is first to set the global[0] to zero then check global[0] again after appl_init call. If global[0] is non zero then AES is present otherwise AES is not installed.

Global[1] will return -1 to indicate current AES supports multitasking environment.

The global[13] contains the current maximum character that is used by AES to do vst_height before writing text to the screen.
Changed 9/29/92

The global[14] contains the current maximum character that is used by AES to do vst_height before writing text to the screen.
Changed 9/29/92

Note:
Some AES bindings have been hard-wired to always return a 1 in int_out[0] instead of the true AES application ID.
Please check all bindings carefully.

Parameters:

control[0] = 10
control[1] = 0
control[2] = 1
control[3] = 0
control[4] = 0

int_out[0] = ap_id

ap_id - If appl_init was successful, ap_id is a zero or a positive number. GEM AES places this number in the Global Array, and the application uses it with future calls to AES routines.

If APPL_INIT was not successful, the value of ap_id is -1. The application should make no further AES calls.

Sample call to C language binding:

```
ap_id = appl_init();
```

3.4.2 APPL READ

Purpose:

Reads a specified number of bytes from a message pipe.

If there is no message in the pipe, AES will change the current application into WAIT state until the request is satisfied.

It is strongly recommended to read 16 bytes at a time because of the way the AES works.

V4.0

If ap_rid equals -1, this function will do a read only if there is data in the message pipe. Otherwise, it will return immediately.

Parameters:

```
control[0] = 11
control[1] = 2
control[2] = 1
control[3] = 1
control[4] = 0

int_in[0]    = ap_rid
int_in[1]    = ap_rlength
int_out[0] = ap_rreturn
addr_in[0] = ap_rpbuff
```

Description:

ap_rid - the ap_id of the process whose message pipe the application is reading (usually its own)

ap_rlength - the number of bytes to read from the message pipe

ap_rreturn - a coded return message

0 - an error exists
n - (positive integer) - no error exists

ap_rpbuff - address of the buffer that will hold the bytes the application is reading

Sample call to C language binding:

```
ap_rreturn = appl_read ( ap_rid, ap_rlength, ap_rpbuff );
```

3.4.9 APPL SEARCH

Purpose:

V4.0
Searches all the existing AES processes in the system.

Parameters:

```
control[0] = 18
control[1] = 1
control[2] = 3
control[3] = 1
control[4] = 0

int_in[0]       = ap_smode
addr_in[0]      = ap_sname
int_out[0]      = ap_sreturn
int_out[1]      = ap_stype
int_out[2]      = ap_sid
```

ap_smode - Search mode of the function
0 = search first (all the processes)
1 = search next (all the processes)
2 = search system shell (only one)

ap_sname - Buffer that will hold the name of the AES process
the size must be 9 or more characters long

ap_stype - Process's type

1 = System process
2 = Application
4 = Accessory

ap_sid - Process's AES id

ap_sreturn - a coded return message

0 - No more file
1 - No error exists

Sample call to C language binding:

```
ap_sreturn = appl_search( ap_smode, ap_sname, &ap_stype, &ap_sid );
```

3.4.10 APPL GETINFO

Purpose:

This function is to allow application to get general information about the AES in general.

Parameters:

```
control[0] = 130
control[1] = 1
control[2] = 5
control[3] = 0
control[4] = 0

int_in[0]       = ap_gtype
int_out[0]      = ap_greturn
int_out[1]      = ap_gout1
int_out[2]      = ap_gout2
int_out[3]      = ap_gout3
int_out[4]      = ap_gout4
```

ap_gtype - Get information type:

0 - Get AES regular font information:
- ap_gout1 - font height
- ap_gout2 - font id
- ap_gout3 - font type: 0 - system font
  1 - FSM font
  2 and on to be defined in the future

1 - Get AES small font information:
- ap_gout1 - font height
- ap_gout2 - font id
- ap_gout3 - font type: 0 - system font
  1 - FSM font
  2 and on to be defined in the future

2 - Get AES current resolution number and the number of color is being supported by the object library.

- ap_gout1 - resolution number
- ap_gout2 - number of color supported by AES object library
- ap_gout3 - color icons: 0 - Not supported
  1 - Supported
- ap_gout4 - new resource file format: 0 - Not supported
  1 - Supported

ap_greturn - a coded return message

1 - No error
0 - Error

Sample call to C language binding:

```
ap_greturn = appl_getinfo( ap_gtype, &ap_gout1, &ap_gout2,
                                                &ap_gout3, &ap_gout4 );
```

8.3.9 GRAF MOUSE

Purpose:

Changes the mouse to one of a predefined set or to an application defined form.
The application should change the mouse back to ARROW after it finishes with its current action. Do not leave the mouse in any form other than ARROW.

V4.0

Currently, applications have free access to the graf_mouse function to change the mouse form without any restriction. However, in the new multiprogramming environment in which a lot of applications may present in the system at the same time will pose a big problem.
Rules will be implemented so that AES can take control of the mouse form and decide which application can change the mouse form and which one can't.

During a normal circumstance, a mouse should always stays in ARROW form. In this case, the mouse form ownership is free and any application can change it at its will. Once the mouse form is being changed to something other than the ARROW, ownership is transfered to that application until it changes the mouse back to ARROW. So, as a courtesy to other applications, the current owner should change the mouse back to ARROW as soon as it finishes with its work.

However, in some circumstances, an application may want to change mouse form immediately without any delay. For example, the foreground application change the mouse to a busy bee and user clicks on the background to do a drag operation on a different application.
It is very logical that the mouse should be changed to a flat hand for the dragging. In this case, AES provides a way to force the current mouse to the next owner in order to deal with this type of situation.
Please read the gr_monumber section below.

Parameters:

```
control[0] = 78
control[1] = 1
control[2] = 1
control[3] = 1
control[4] = 0

 int_in[0] = gr_monumber
int_out[0] = gr_moreturn
addr_in[0] = gr_mofaddr
```

gr_monumber - a code identifying a predefined form

0 - arrow
1 - text cursor
2 - busy bee
3 - hand with pointing finger
4 - flat hand, extended fingers
5 - thin cross hair

6 - thich cross hair
7 - outline cross hair
255 - mouse form stored in gr_mofaddr
256 - hide mouse form
257 - show mouse form
(V4.0)
258 - save current mouse form
259 - restore to the last saved mouse form
260 - restore to previous mouse form

In the event that the application must change the mouse form, set the highest bit (Bit 15) of gr_monumber to 1 and OR in with the desired mouse form number. After finishing the work, call the graf_mouse with value 0 to set the mouse back to arrow. It is suggested that the application to make this call in the following way:

```
wind_update( 1 )
 graf_mouse( 0x8000|gr_monumber, 0 );
 actions()
 graf_mouse( 0, 0 );
wind_update( 0 )
```

gr_moreturn - a coded return message

0 - an error exits
n (positive integer) - no error exists.

gr_mofaddr - the address of a 35-word buffer that fits the mouse form definition block specified in the GEM programmer's guide

Sample call to C language binding:

```
gr_moreturn = graf_mouse( gr_monumber, gr_mofaddr );
```

Version Documentation Date:
September 9, 1992

Section 5 Addendum

Menu Library Enhancements

5.1 Introduction

This section describes the additional features of the Menu Library. All enhancements are backwards compatible with previous versions of the AES, so existing applications will continue to work. The new features will work on all machines with an AES version number of 3.3 and up.

The enhancements to the Menu Library are:

- o Hierarchical menus are now supported.
- o Pop-Up Menus are now supported.
- o Scrolling menus are supported for pop-up menus and submenus. Scrolling for the first level menus of a menu bar are not supported.

Hierarchical Menus

Hierarchical menus allow a menu item to be the title of a submenu. Menu items with a right arrow signify that a submenu is attached. Heirarchical menu items must be of the type G_STRING. As a rule, the Desk Menu of a menu bar is not allowed to have submenus.

Two delay values are used to prevent the rapid appearance and disappearance of submenus:

- o **Submenu Display Delay**

  This delay is used to prevent the rapid flashing of submenus as the mouse pointer is dragged thru a menu item with an attached submenu. The mouse pointer must remain within the menu item for the delay period before the submenu is displayed.
  The default Submenu Display Delay is 1/5 of a second.
  menu_settings can be used to inquire the current delay value, or to set a new delay.

- o **Submenu Drag Delay**

  This delay is used to prevent the disappearance of the submenu as the mouse pointer is dragged toward the submenu from a menu item.
  The default Submenu Drag Delay is 10 seconds. menu_settings can be used to inquire the current delay value, or to set a new delay.

There are several actions that will cancel the Submenu Drag Delay prematurely:

1) If the mouse pointer is dragged away from the direction of the submenu, the submenu will disappear.

2) If the mouse pointer remains in the same position after the drag has begun, the submenu will also disappear.

3) If the user clicks on the left mouse button before the mouse pointer has entered the submenu, the system will return to the application the menu item that started the drag.

4) If the mouse pointer is dragged vertically into another menu item, the submenu will disappear.

As a rule, only one level of heirarchical menus should be used. The actual number of recursions possible is currently set to 4.

Pop-Up Menus

Pop-up menus are menus that are not in the menu bar. They can be placed anywhere on the screen and once displayed, act like any other menu.

Scrolling Menus

When the number of menu items exceeds the menu scroll height, a scroll indicator appears at the bottom of the menu. The scroll indicators are displayed as UP or DOWN ARROWS. Clicking on the bottom arrow will scroll the menu items. When the last item is shown, the DOWN ARROW indicator disappears. Note that as soon as the menu started scrolling, the UP ARROW indicator appeared at the top of the menu. This is to show that there are now menu items in that direction. The default menu scroll height is 16. menu_settings can be used to inquire the current menu scroll height, or to set a new menu scroll height.

When the user clicks and holds down the left mouse button, there is a 1/4 of a second delay after one menu item has scrolled. After the delay, scrolling continues uninterrupted. This delay is used to prevent rapid scrolling for those just clicking on the scroll indicators. menu_settings can be used to inquire the current delay, or to set a new delay.

Another delay value is used to slow down the scrolling speed. This prevents the menu items from scrolling by too fast. menu_settings can be used to inquire the current delay, or to set a new delay.

Pop-up menus and submenus might consist of objects other than G_STRINGS. Such a menu might consist of user-defined objects that display the system's fill patterns. The system cannot scroll non-G_STRING object types. Scrolling non-G_STRING object types will crash the system, Pop-up menus and submenus containing non-G_STRING object types should have its scroll_flag field set toFALSE.

The first-level menus of a menu bar are set to be non-scrollable. This is due to the parent-child relationships between the menu titles, menus and menu items. Therefore, scrolling is applicable only to pop-up menus and submenus.

5.2 Using the Extended Menu Library

The existing Menu Library functions are still applicable to pop-up menus and submenus. The Menu Library will continue to have the following responsibilities:

- o displaying the appropriate menu bar for each active application
- o enabling and disabling menu items
- o displaying check marks in menus
- o returning a highlighted menu title to its normal state
- o displaying context-sensitive menu text
- o displaying a desk accessory's name on the Desk Menu

To use pop-up menus and submenus in one's application:

Create an object tree consisting of a G_BOX and as many G_STRINGS within the G_BOX as required. The G_BOX is the menu and the G_STRINGS are the menu items. An object tree is not limited to just one menu and can contain one, two or more menus. If a menu item is expected to have a submenu attachment, the G_STRING must be padded with blanks to the width of the menu.

The object tree does not need to be created with the Resource Construction Set. It can be created during runtime by the application. However, the programmer is responsible for this procedure.

Attaching a submenu to a menu item is done by calling menu_attach. A submenu is associated to a menu item by placing a right arrow two characters in from the right edge. Any characters at that location will be overwritten.

In addition, the high-byte of the object's type field is used to store an internal Menu ID. The values between 128 and 192 are used by the new menu sytem. Value one (1) and two (2), are used by a new AES feature. Currently, the resource contruction program - INTERFACE, uses the values 17 through 22 for their library routines. The Extended Object Type field is currently under evaluation at Atari and a forthcoming document will describe which values are reserved for Atari's internal use only, and which values are available for application purposes.

Each process can have up to 64 unique submenu attachments. Attaching the same submenu to multiple menu items counts as one attachment.

In addition to attaching a submenu, menu_attach can be used to change or remove a submenu. menu_attach can also be used to find out what submenu, if any, is attached to a menu item. menu_istart can be used to set and get the starting menu item of a submenu.

menu_settings can be used to set the menu delay values and to set the height at which pop-up menus and submenus will start to scroll.

5.2.1 Using a Menu Bar

Supplement to 5.2 Using the Menu Library

When the user chooses an item, the Screen Manager writes a message to the pipe. Control then returns to the application, which must read the pipe.

The pipe message contains the following:

- +0 o a code indicating that it is a menu message (MN_SELECTED)
- +6 o the object index of the menu title selected
- +8 o the object index of the menu item chosen
- +10 o the object tree of the menu item chosen (NEW)
- +14 o the object index of the parent of the menu item (NEW)

(If the user does not choose an item, or if the user selects a disabled menu item, the Screen Manager does not write a message to the pipe.)

After processing the chosen item, the application makes a Menu Library call to dehighlight the menu title and waits for the next message to come through the message pipe.

5.3-b Extended Menu Library Routines

The additions to the Menu Library routines are:

- menu_popup: Displays a menu anywhere on the screen. Clipping is performed for a standard menu. Menus with user-defined objects will have to perform their own clipping.
- menu_attach: Lets an application attach, change, remove or inquire about a submenu associated with a menu item.
- menu_istart: Lets an application set and inquire the starting menu item of a pop-up menu or submenu
- menu_settings: Lets an application set and inquire the delay and height parameters of the submenus.

menu_popup and menu_attach use a new structure for passing and receiving submenu data. The MENU structure is defined as follows:

```
typedef struct _menu
{
   OBJECT *mn_tree;    - the object tree of the menu
   WORD    mn_menu;    - the parent object of the menu items
   WORD    mn_item;    - the starting menu item
   WORD    mn_scroll;  - the scroll field status of the menu
                         0  - The menu will not scroll
                         !0 - The menu will scroll if the number of menu
                              items exceed the menu scroll height. The
                         non-zero value is the object at which
                         scrolling will begin.      This will allow one
                         to have a menu in which the scrollable region
                         is only a part of the whole menu.      The value
                         must be a menu item in the menu.

                              menu_settings can be used to change the menu
                              scroll height.

                         NOTE: If the scroll field status is !0, the menu
                               items must consist entirely of G_STRINGS.
   WORD    mn_keystate; - The CTRL, ALT, SHIFT Key state at the time the
                          mouse button was pressed.
}MENU;
```

menu_settings uses a new structure for setting and inquiring the submenu delay values and the menu scroll height. The delay values are measured in milliseconds and the height is based upon the number of menu items.

```
typedef struct _mn_set
{
   LONG  Display; - the submenu display delay
   LONG  Drag;    - the submenu drag delay
   LONG  Delay;   - the single-click scroll delay
   LONG  Speed;   - the continuous scroll delay
   WORD  Height;  - the menu scroll height
}MN_SET;
```

- Submenu Display Delay:

  The delay is used to prevent the rapid flashing of submenus as the mouse pointer is dragged thru a menu item with an attach submenu. The default value is 200 milliseconds (1/5th of a second).

- Submenu Drag Delay:

  The delay is used to prevent the disappearance of the submenu as the mouse pointer is dragged toward the submenu from a menu item. The default value is 10000 milliseconds (10 seconds).

- **Single-Click Scroll Delay:**

  This is the delay period after one menu item has initiallly scrolled. After the delay, scrolling continues at the rate specified by the Continuous Scroll Delay. The delay is used to prevent rapid scrolling for those just clicking on the scroll indicators. The default value is 250 milliseconds (1/4th of a second).

- **Continuous Scroll Delay:**

  This is the delay period after each menu item has scrolled. The delay is used to slow down the scrolling speed.
  The default value is 0 milliseconds.

- **Menu Scroll Height:**

  This value is the height at which a pop-up menu or a submenu will start to scroll if its scroll field is TRUE.
  The default value is 16 menu items.

The following sections describe these routines.

5.3.7 MENU POPUP

Purpose:

Allows an application to display a popup menu anywhere on the screen. The popup menu may also have submenus. If the number of menu items exceed the menu scroll height, the menu may also be set to scroll. menu_settings can be used to set the height at which all menus will start to scroll.

Parameters:

```
control(0)    =  36
control(1)    =  2
control(2)    =  1
control(3)    =  2
control(4)    =  0

int_in(0)     =  me_xpos
int_in(1)     =  me_ypos

int_out(0)    =  me_return

addr_in(0)    =  me_menu
addr_in(1)    =  me_mdata
```

me_xpos - the left edge of where the starting menu item will be displayed

me_ypos - the top edge of where the starting menu item will be displayed

me_return - a coded return message

```
0 - FAILURE: The data returned by me_mdata is invalid
1 - SUCCESS: The data returned by me_data is valid
```

FAILURE is returned if the user did not click on an enabled menu item

me_menu - pointer to the pop-up MENU structure. The structure must be initialized with the object tree of the pop-up menu, the menu object, the starting menu item and the scroll field status.

e_mdata - pointer to the data MENU structure. If menu_popup returns TRUE, me_mdata will contain information about the submenu that the user selected. This includes the object tree of the submenu, the menu object, the menu item selected and the scroll field status for this submenu.

Sample call to C language binding:

```
me_return = menu_popup( MENU *me_menu, word me_xpos, word me_ypos,
                        MENU *me_mdata);
```

5.3.8 MENU ATTACH

Purpose:

Allows an application to attach, change, remove or inquire about a submenu associated with a menu item.

Parameters:

```
control(0)     =  37
control(1)     =  2
control(2)     =  1
control(3)     =  2
control(4)     =  0

int_in(0)      =  me_flag
int_in(1)      =  me_item

int_out(0)     =  me_return

addr_in(0)     =  me_tree
addr_in(1)     =  me_mdata
```

me_flag - the action to be performed by menu_attach.

The options for me_flag are:

0 Inquire data about the submenu that is associated with the menu item. The data concerning the submenu is returned in me_mdata.

1 Attach or change a submenu associated with a menu item. me_mdata must be initialized by the application. The data must consist of the object tree of the submenu, the menu object, the starting menu item and the scroll field status.
Attaching a NULLPTR structure will remove the submenu associated with the menu item. There can be a maximum of 64 associations per process.

2 Remove a submenu associated with a menu item. me_mdata should be set to NULLPTR.

me_item - the menu item that the submenu will be attached to

me_return - a coded return message

0 - FAILURE: the submenu was not attached for whatever reasons
1 - SUCCESS: the submenu was attached, changed or removed successfully

me_tree - the object tree of the menu item that will have a submenu attach to

me_mdata - pointer to the MENU structure. The contents of me_mdata are dependant upon the value of me_flag:

0 Upon return from menu_attach, me_mdata will contain the MENU data regarding the submenu associated with the menu item.

1 me_mdata must be initialized with the new submenu MENU data. The submenu will be attached to the menu item - me_item.

2 me_mdata should be set to NULLPTR. The submenu associated with the menu item will be removed.

Sample call to C language binding:

```
me_return = menu_attach( word me_flag, object *me_tree, word me_item,
                         MENU *me_mdata );
```

5.3.9 MENU ISTART

Purpose:

Allows an application to set or inquire the starting menu item of a submenu that is associated with a menu item. The submenu is shifted vertically so that the starting menu item is aligned with the menu item that is associated with this submenu.

Parameters:

```
control(0)    =  38
control(1)    =  3
control(2)    =  1
control(3)    =  1
control(4)    =  0

int_in(0)     =  me_flag
int_in(1)     =  me_imenu
int_in(2)     =  me_item

int_out(0)    =  me_return

addr_in(0)    =  me_tree
```

me_flag - the action to be performed by menu_istart

0 Inquire the starting menu item for the submenu

1 Set the starting menu item for the submenu to be me_item

me_imenu - the menu object of the submenu that is either to be set or inquired

me_item - the starting menu item that is either to be set or inquired

me_return - a coded return message

0 - FAILURE: the submenu is not associated with a menu item. The submenu must be attached via menu_attach before this call can be made.

>0 - SUCCESS: the starting menu item is currently set to this value.

me_tree - the object tree of the menu item that we are setting or inquiring about

Sample call to C language binding:

```
me_return = menu_istart( word me_flag, object *me_tree, word me_imenu,
                         word me_item );
```

5.3.10 MENU SETTINGS

Purpose:

Allows an application to set or inquire the submenu delay values and the menu scroll height value.

Parameters:

```
control(0)      =  39
control(1)      =  1
control(2)      =  1
control(3)      =  1
control(4)      =  0

int_in(0)       =  me_flag

int_out(0)      =  me_return

addr_in(0)      =  me_values
```

me_flag - the action to be taken by menu_settings

- 0 Inquire the current delay and menu scroll height values.
- 1 Set the delay and menu scroll height values

me_return - always returns 1 (one)

me_values - pointer to the MN_SET structure. me_values is dependant upon the value of me_flag:

- 0 Upon the return of menu_settings, me_values will contain the current delay and menu scroll height values.
- 1 me_values must be initialized. The delay and menu scroll height values will be set to those values found in me_values. A value set to NIL will be ignored.

Sample call to C language binding:

```
me_return = menu_settings( word me_flag, MN_SET *me_values );
```

AES Supplemental Documentation

The following section contains documentation supplemental to the existing AES manual, and clarifications of existing documentation related to heirarchical submenus and the menubar.

Supplement to: 4.2.5.2 MN SELECTED

GEM AES uses this message to notify an application that a user has selected a menu item.

- word 0 = 10
- word 3 = the object index of the menu title selected
- word 4 = the object index of the menu item selected
- word 5,6 = the object tree of the menu item selected
- word 7 = the parent object of the menu item selected

5.4.1 Pop-Up Menus

- The button on a dialog box that brings up a pop-up menu should be shadowed.
- It would be nice if the pop-up menu was shadowed also.
- While the pop-up menu is displayed, if it has a title, the title shoul be inverted.
- The pop-up menu should be aligned on a byte boundary. This speeds up the drawing of the menu considerably.
- The pop-up menu will be shifted vertically in order to line up the start object with the given coordinates.
- If the menu exceeds the top of the screen, it will be shifted down.
- No horizontal adjustments will be done to the menu.

5.4.2 Submenus

- Menu items expecting a submenu attachment must be of type G_STRING.
- Menu items should be padded with blanks to the width of the menu.
- Menu items expecting a submenu attachment should not have any keyboard short-cut characters.
- Submenus will automatically be displayed on a byte boundary.
- The menu will be shifted vertically to align the start object with the menu item. In addition, the menu will be shifted to remain entirely on the screen in the vertical plane.
- The submenu will be displayed at the right edge of the menu item. If the menu extends off the edge of the screen, the menu will be displayed to the left of the menu item.
  If it exceeds the left edge of the screen, the menu will be shifted right a character at a time, until it fits.
- There can be a maximum of 64 submenu attachments per process.
- A menu item with an attached submenu uses the high-byte of its object type field. Values 128 thru 192 are used by the submenu menu system.

  Value one (1) and two (2), are used by a new AES feature.

  Please note that the Interface program already uses values 17 thru 22 for its library routines. The Extended Object Type field is currently under evaluation at Atari. A forthcoming document will describe which values are reserved for Atari's internal use and which are available for application purposes.
- A submenu should not be attached to itself.
- Attaching a submenu to different menu items counts as one attachment. There will only be one scroll flag and one start object.
- As a user interface guideline, there should only be one level of hierarchical menus. The system currently allows up to four levels of recursion.
- menu_istart works only on submenus attached with menu_attach.

5.4.3 Scrolling Menus

- In order to scroll properly, all menu items must be G_STRINGS. Menus that contain objects other than G_STRINGS should set the scroll flag to 0.
- The first-level menus of a menu bar are not scrollable.
- Pop-up menus and submenus with greater than sixteen items can be scrolled if their scroll flag is set. The number of items to scroll at can be adjusted with menu_settings.
- If the pop-up menu or submenu is designed to be a toolbox, (ie: fill patterns), set the scroll flag to FALSE.
- Setting the scroll flag to one of the menu items will initiate scrolling from that menu item if the number of items exceeds the menu height limit.
- One should NOT set the scroll object to the last menu item of a menu.
- Setting the scroll object to a value less than the first menu item defaults to the first menu item.
- Setting the scroll object to a value greater than or equal to the last menu item defaults to the first menu item.

5.3.1 MENU BAR

Purpose:

Displays or erases the application's menu bar

The application should always call MENU_BAR to erase the menu prior to its APPL_EXIT call.

V4.0

If me_bshow is -1, the menu_bar will become an inquiry call in which it will return the current menu owner's AES process id. If the return value is -1, then there is no menu bar owner.

It is important to point out that the current menu bar can be swappped out at any time. If the application wants to update or redraw its menu bar, it is recommended to first check to see if it still own the menu bar and then proceed to its functions. However, the menu bar owner can still be changed after menu_bar call. So the safest way to do menu bar update function should be as follow:

```
wind_update(1);          /* wait until the screen stable down      */
                         /* no nobody can change the menu          */

id = menu_bar( 0x0L, -1 );

if ( id == my_id )
{
   /* update the menu and draw it */
}
else
{
   /* update the menu but don't draw it */

}

wind_update(0);
```

Parameters:

```
control[0] = 30
control[1] = 1
control[2] = 1
control[3] = 1
control[4] = 0

int_in[0]  = me_bshow
int_out[0] = me_breturn
addr_in[0] = me_btree
```

me_bshow - a code for whether the application displays the menu bar.

0 - erase the menu bar
1 - display the menu bar

V4.0

-1 - Inquire current menu owner.

me_breturn - a coded return message.

0 - an error exists
n (positive integer) - no error exists

For inquiry mode, -1 indicates no menu owner, otherwise n is the current menu owner's AES id.

me_btree - the address of the object tree that forms this menu.

Sample call to C language binding:

```
me_breturn = menu_bar( me_btree, me_bshow );
```

5.3.6 MENU REGISTER

Purpose:

Places a desk accessory's menu item string on the menu and returns the accessory's menu item identifier.

The menu can list no more than six desk accessories.

V4.0

The AES can handle as many accessories as possible provided that the drop-down menu is tall enough to hold all the accessories' name or that is enough memory to load in all the accessories.

Applications can call menu_register to change the name that appears in the menu bar for that application. The parameters are the same.

Parameters:

```
control[0] = 35
control[1] = 1
control[2] = 1
control[3] = 1
control[4] = 0

int_in[0] = me_rapid
int_out[0] = me_rmenuid
addr_in[0] = me_rpstring
```

Description:

me_rapid - The AES process identifier of the desk accessory or application. This value is the ap_id returned by the appl_init call.

If this value equals -1, the me_rpstring will be used to replace the current accessory's process name.

me_rmenuid - The desk accessory's or application's menu item identifier.

-1 - no more room on the menu.

me_rpstring - The address of the desk accessory's or application's menu text string. This string must contain no more than 8 characters and must be ended with a dot or null.

Sample call to C language binding:

```
me_rmenuid = menu_register( me_rapid, me_rpstring );
```

12.3.6 RSRC RCFIX

Purpose:

V4.0

Fixes up a raw resource data that is already loaded into the memory by the application.

- It converts all the object's locations and sizes into pixel coordinates.
- The resource must be the same as those generated by the resource construction set.
- If there is another resource already loaded into the system for the application, the application is required to do a rsrc_free to free up the memory before calling this function.
- Application still needs to do rsrc_free before terminate.

Parameters:

control[0] = 115
control[1] = 0
control[2] = 1
control[3] = 1
control[4] = 0

int_out[0] = rc_return
addr_in[0] = rc_header

rc_return = Return value, always equals to 1
rc_header = Resource header memory location. It must be followed by resource data.

Sample call to C language binding:

```
rc_return = rsrc_rcfix( rc_header );
```

13.3.5 SHEL_GET

Purpose:

Lets an application read data from the AES's internal shell buffer.

The length of the buffer should not be more than 1024 bytes.

V4.0

sh_greturn - returns the size of the data. The data can be longer than 1024 bytes. The application's buffer will not be filled with more than sh_glen bytes, even if there is more data in AES's internal buffer.

5/5/92
If sh_glen is -1, it is the inquiry mode.

Parameters:

control(0) = 122
control(1) = 1
control(2) = 1
control(3) = 1
control(4) = 0

int_in(0) = sh_glen

addr_in(0) = sh_gbuff

int_out(0) = sh_greturn

Description:

sh_greturn - a coded return message

0 - an error exists
n (positive integer) - no error exists

sh_glen - the length of the buffer.

If the value is -1, it is the inquiry mode in which sh_greturn will return the size of the current shel_get buffer.

sh_gbuff - the address of the buffer.

Sample call to C language binding:

```
sh_greturn = shel_get( sh_gbuff, sh_glen );
```

13.3.6 SHEL PUT

Purpose:

Lets the application save data into the AES's shell internal buffer.

Note:
Currently, the AES desktop is using this buffer to store the desktop.inf data. Any usage of this buffer may corrupt the data that are already stored in there.

The length of the data that goes into the buffer should not be more than 1024 bytes.

V4.0

The AES will allocate memory to store the data if the current buffer size is smaller than the sh_plen size. The default buffer size is 1024 bytes and the maximum size must be be less than 32K bytes.

Parameters:

control(0) = 123
control(1) = 1
control(2) = 1
control(3) = 1
control(4) = 0

int_in(0) = sh_plen

addr_in(0) = sh_pbuff

int_out(0) = sh_preturn

sh_preturn - a coded return message

0 - an error exists
n (positive integer) - no error exists

sh_plen - the length of the buffer.

sh_pbuff - the address of the buffer.

Sample call to C language binding:

```
sh_preturn = shel_put( sh_pbuff, sh_plen );
```

Update 5/7/92

Changes in mode 4, 5, and 7

New shel write mode 9 and 10

13.3.2 SHEL WRITE

Purpose:

Tells GEM AES whether to run another application and, if so, which application to run.

```
sh_wreturn = shel_write( sh_wdoex, sh_wisgr, sh_wiscr,
                         sh_wpcmd, sh_wptail );
```

V4.0

Shel_write is expanded to have multiple functionalities. Please study the following documentation carefully.

-> Sh_wodex:
- 0 - Launch program. The actual sh_wisgr value will be determined by the AES.
- 1 - Launch an application
- 3 - Launch an accessory.
- (value 2 is reserved)

Default directory will be set according to the file's full path except for the extended mode if bit 10 is set.

AES will create a process of that applibation and will not wait for it to terminate (i.e. the process is started concurrently).

sh_wreturn will return the process's AES id. If it is 0, an error exists.

If sh_wodex is 0, the AES will determine the actual launching mode by looking at the file's extension. What file extensions are considered for launching is determined by the AES environment variables GEMEXT, TOSEXT, and ACCEXT.

-> Sh_wisgr: This parameter is only valid when Sh_wodex is 1

- 1 for GEM application
- 0 for TOS application

The launching a non-gem (TOS) type application by shel_write is implemented in the following way:

The AES will look into the environment for a variable named TOSRUN. It should contain a full path of a tos handler program to which the AES will pass the current (TOS) program name into the command tail. If there is an error, of launching this tos handler program, AES will return immediately.
If TOSRUN does not exist, the AES will use the default way (MW.PRG's message pipe method) to launch that program.

For example:

Launch a program named C:\E.TOS and the current
TOSRUN=C:\BIN\TOSHAND.PRG

The following action will take place:

The TOSHAND.PRG will be launched, the command tail will be preserved. However, an ARGV= will be created in the environment in the following way.

```
ARGV=\0
C:\E.TOS\0
file1\0 ( If there is any )
file2\0 ( If there is any )
\0
```

Note:

The TOSHAND.PRG will be launched as a GEM program regardless of its real type. The PATH will not be used to search the program so that TOSHAND.PRG can use TOSRUN to locate where it is.

-> Sh_wiscr: Request AES for setting up ARGV style parameter passing in the environment string
0 - No
1 - Yes

This feature is intended for application which passes one or more arguments in the command tail.

The AES will extract arguments from the command tail (seperated by spaces) and construct an ARGV element (as the last element) in the environment in the following fashion:

```
ARGV=\0
Progam name\0
Argument1\0
Argument2\0
Argument3\0
\0
```

A value of 0x7f will be put into the first byte of original command tail to indicate a valid ARGV presents in the environment, application should ignore the content in the command tail. 5/22/92

See The Atari Extended Argument Specification (ARGV), a separate document.

More special features (Extended mode):

The shel_write call also allows the user to launch the program or accessory in a customized way. The high byte (bit 15 to bit 8) of sh_wodex is checked to see if any of the bits is set. If so, it is an extended call. Each bit is assigned to have a special meaning, the low byte meaning is not affected.

SH_WODEX

```
              High Byte        |     Low Byte
Bit Number: 15 14 13 12 11 10 9 8 | 7 6 5 4 3 2 1 0
     ---------------------------------------------
                          No Change
 8 -> Psetlimit value
 9 -> Prenice value
10 -> Default directory string
11 -> Environment string
12   -> Reserved
13 -> Reserved
14 -> Reserved
15 -> Reserved
```

In extended mode, sh_wpcmd will be treated as a pointer pointing to a set of long (32 bit) values. Each value after the first corresponds to one of the bits in sh_wodex: if that bit is set then the corresponding LONG value is used, otherwise it is ignored. The values and their associated bit numbers are as follows:

a, LONG[0] Pointer to the program name string
(must be the first element)
b, LONG[1] Psetlimit value. (bit 8)
c, LONG[2] Prenice value. (bit 9)
d, LONG[3] Default directory string pointer (bit 10)

The directory path (LONG[3]) should look something like:

C:\ or C:\FOLDER or C:\FOLDER1\FOLDER2 ...

However, if the pointer is zero, then the default directory will be the directory that the program itself was found in.

e, LONG[4] Application defined environment string pointer (bit 11)

PATH SEARCHING

When using sh_wodex mode 0, 1, and 3, the program name string (LONG[0]) may contain full path and file name specifying the file to launch, or it may contain simply the name of the program. In that case, AES will search each directory in the PATH variable in AES' environment for a file with that name and a legal program extension, as specified in the TOSEXT, GEMEXT, and ACCEXT environment variables.

In mode 0 and mode 3, ACCPATH variable will be searched also.

If the program is found using one of the GEMEXT extensions, it is launched as a GEM application. If it is found using one of the TOSEXT extensions, it is launched like TOS programs are (see elsewhere). If it is found using one of the ACCEXT extentions, it is loaded as a desk accessory.

If the program name string includes an extension, that extension is checked against those in GEMEXT, TOSEXT, and ACCEXT to determine how to launch the program. If the extension is not found in any of those environment variables, an error is returned.

If the supplied program name includes any directory separators (backslashes) or drive specifiers (a letter and a colon) then the path and extension searching will not be done. If there are no drive or directory specifiers but there is an extension (a dot), then the path searching is done but the extension searching is not. This is illustrated in the following table:

PATH NAME	PATH SEARCH	EXT SEARCH
C:\BIN\MW.PRG	No	No
C:\BIN\MW	No	Yes 4/27/92
C:\BIN\MW.	No	No
\MW.PRG	No	No
MW.PRG	Yes	No
MW.	Yes	No
MW	Yes	Yes

sh_wodex mode 1 explicitly calls for the program to be launched as a GEM application or a TOS application (based on sh_wisgr). Mode 3 explicitly calls for the file to be loaded as an accessory. Mode 0 causes AES to decide, based on the extension.

-> Sh_wodex value 4

Set the system in shutdown or normal mode depends on sh_wiscr value.

Once the AES is in the shutdown mode, the shel_write launch file capability mode 0-2 will be turned off.

sh_wisgr 2 - Complete shutdown mode

AES will check for all applications and accessories excluding the caller to make sure they all recognize AP_TERM message. If succeeded, AES will then send out AP_TERM to applications and AC_CLOSE to accessories. Accessories also get AP_TERM after the AC_CLOSE message.

Note: the caller will receive none of the messages.

sh_wisgr 1 - Partial shutdowm mode

AES will check for all applications excluding the caller to make sure they all recognize AP_TERM message. If succeeded, AES will then send out AP_TERM to applications and AC_CLOSE to accessories.

Note: the caller will receive none of the messages.

sh_wisgr 0 - Abort the shutdown sequence.

Note: Only the original caller of shutdown mode can abort the shutdown sequence.

Please see shel_write mode 9 of how an application informs AES that it can recognize the AP_TERM message.

-> Sh_wodex value 5 (Changed 9/29/92)

Request the AES to change resolution. The sh_wiscr's value affects the meaning of sh_wisgr. If AES accepts the resolution change request, then it will put the system in SHUT DOWN mode. An application can either shut down and exit or deny to shut down by sending a AP_TFAIL message to the AES.

a, If sh_wiscr is zero, then sh_wisgr is the physical device id to perform VDI's open physical workstation call.

To get the current physical device id, do the Getres()+2.

The following are the existing physical device id values:

2 -> 320x200	ST Low
3 -> 640x200	ST Medium
4 -> 640x400	ST High
6 -> 640x480	TT Medium
8 -> 1280x960	TT High
9 -> 320x480	TT Low

b, If sh_wiscr is one, then sh_wisgr is the video mode word for use in the FALCON030 machine.

Sh_wiscr value from 2 and up are reserved for future use.

-> Sh_wodex value 7

Sends a message (broadcast) to all processes except AES,SCREEN and the sender.

In this mode, the sh_wpcmd will be treated as a pointer pointing to a 16 byte message buffer. Sh_wisgr and sh_wiscr are ignored.

-> Sh_wodex value 8

This shel_write mode allows applications to manipulate AES environment variables.

1. sh_wisgr = 0 Inquire the environment buffer size

   Sh_wreturn returns the size in bytes

2. sh_wisgr = 1 Add/Remove string

   In this mode to add or remove an AES environment name.

   The sh_wpcmd is the new environment string.

   a. To add a new environemnt string, the input should look like 'NEW=STRING\0'

   b. To remove an environment string, the input should look like 'NEW=\0'

3. sh_wisgr = 2 Copy the environment buffer

   The sh_wpcmd is the output buffer in which the AES will copy the current environment buffer to.

   The output buffer size is specified by sh_wiscr.

   Sh_wreturn returns the number of bytes not copied.

-> Sh_wodex value 9

Inform the AES of what kind of new message that the application can recognize.

Sh_wisgr is the input parameter in which each of the 16 bits represents a message type.

Bit 0 - AP_TERM
Bit 1-15 Not defined at this moment.

-> Sh_wodex value 10

Send the AES a message.
Sh_wpcmd is the 16 byte message buffer

Parameters:

control(0) = 121
control(1) = 3
control(2) = 1
control(3) = 2
control(4) = 0

int_in(0) = sh_wdoex
int_in(1) = sh_wisgr
int_in(2) = sh_wiscr

int_out(0) = sh_wreturn

addr_in(0) = sh_wpcmd
addr_in(1) = sh_wptail

Description:

sh_wdoex - a coded instruction to exit the system or run another application when the user exits the current application.

0 - run application
1 - run another application in GEM or TOS mode
3 - run an accessory
4 - set shutdown mode
5 - resolution change
7 - send message to all processes
8 - AES environment

sh_wisgr - function depneds on the sh_wdoex value.

sh_wiscr - function depends on the sh_wdoex value.

sh_wreturn - return value depends on the sh_wdoex function.

sh_wpcmd - the address of the new command file to execute

sh_wptail - the address of the command tail for the next program

The first byte is the length of the command tail, the actual command tail should start from the second byte position in the buffer and should be NULL terminated.
(11/03/92) In V4.0, however, if the first byte has a value of 0xFF, the length of the buffer will be determined by AES by looking for the NULL character. In theory, application can pass more than 128 or 256 bytes to AES to create a long ARGV enviornment by setting sh_wiscr to one.

Sample call to C language binding:

```
sh_wreturn = shel_write( sh_wdoex, sh_wisgr, sh_wiscr,
                         sh_wpcmd, sh_wptail );
```

Supplement to

wind_get / wind_set functions

6/23/92

The window redraw message has been greatly optimized. The changes are in the WM_TOP and WM_SIZE of the window messages. For WM_TOP, there will be no redraw message to follow if the window is not being covered, otherwise, the covered areas will be merged to form a big redraw rectangle in the next WM_REDRAWmessage.

For WM_SIZE, no redraw message will be sent if the window is sized from bigto small and the x and y coordinates remain the same, otherwise, message will be sent.

11.3.5 WIND GET / WIND SET

wi_gfield - A numerical value describing the wind_get operation

WF_TOP - 10 Version 3.3 and up

Returns handle in wi_gw1, owner's AES id in wi_gw2 and the handle of the window below it in wi_gw3

WF_NEWDESK - 14 Version 3.3 and up

Get the current system background window's object pointer. The value is returned in wi_gw1 and wi_gw2.

WF_COLOR - 18 Version 3.3 and up changed 7/8/92

Get the window's element color by handle. The value are returned in wi_wg2 and wi_wg3. See also WF_DCOLOR.

WF_DCOLOR - 19 Version 3.3 and up changed 7/8/92

Get the default element color. The values are returned in wi_gw2 and wi_gw3.

Note: The binding for wind_get for getting the window's color elements is as follows:
wind_get(wi_handle, wi_field, &wi_gw1, &wi_gw2, &wi_gw3, &wi_gw4);
where wi_gw1 should contain the element number as an input, the output values will be placed in the wi_gw2 and wi_gw3.

The wi_gw1 value should be put in the intin[2].

WF_OWNER - 20 Version 3.3 and up changed 7/8/92

Get the window owner's AES id, window open status.

wi_gw1 returns AES id of the owner
wi_gw2 returns if the window is opened (1) or closed (0)
wi_gw3 handle of the window above it
wi_gw4 handle of the window below it

Note: If the window is closed, the wi_gw3 and wi_gw4 values will be meaningless.

WF_BEVENT - 24 Version 3.31 and up changed 7/8/92

If you set bit zero of the WF_BEVENT attribute of a window to one, a button click in that window's work area will not cause a WM_TOPPED message to be sent to that window. Instead, the button click will satisfy evnt_button or the button-click option of an evnt_multi call.

Example:

To turn on this feature for a particular window, set bit 0 of wi_sw1 to 1.
wind_set(wi_handle, WF_BEVENT, 0x0001, 0, 0, 0);

To turn off this feature for a particular window, set bit 0 of wi_sw1 to 0.
wind_set(wi_handle, WF_BEVENT, 0x0000, 0, 0, 0);

To inquire the WF_BEVENT status of a particular window:
wind_get(wi_handle, WF_BEVENT, &wi_sw1,&wi_sw2,&wi_sw3,&wi_sw4);

The return value is in the wi_sw1.

Note:
The rest of the bits and parameters are reserved for future improvement and they should all be zero!

WF_BOTTOM - 25 Version 3.31 and up

This function mode set an already opened window to the bottom of the window stack (excluding the background window) and bring up next logical window to top. However, if the target window is only opend window in the system, this window will still remain on top and be active.

Example:

To set a particular window to the bottom.
wind_set(wi_handle, WF_BOTTOM, 0, 0, 0, 0);

To inquire the current bottom window handle.
wind_get(wi_handle, WF_BEVENT, &wi_sw1,&wi_sw2,&wi_sw3,&wi_sw4);

The handle will return in the wi_sw1;

11.3.8 WIND_UPDATE

V4.0

The new 'check and set mode' is defined as follow.

If the wi_ubegend value is 257 (BEG_UPDATE|0x0100) or 259 (BEG_MCTRL|0x0100). The AES will first check for current wind_update ownership. If nobody owns it or the current owner is the caller itself then normal wind_update procedure will be performed. Otherwise, wind_update will return an error (0) in wi_ureturn.

NEW PREDEFINED AES MESSAGES

The following are the new predefined system messages implemented in the new AES:

1. WM_UNTOPPED - 30 Version 3.3 and up

word 0 - 30
word 3 - window handle

This message is sent when the current top window is being untopped by other window.

The application doesn't need to take any action. This message is for your information only.

By the time this message is received by the owner of the formerly-topped window, that window is not likely to be on top.

2. WM_ONTOP - 31 Version 3.3 and up

word 0 - 31
word 3 - window handle

This message is sent when the application's window is placed on top, generally through no action of its own (i.e. another window is closed). The application does not have to take any action.

Note: Since a lot of window events can happen between the time the window managers send out this message and the time the application actually receives the message, the window in question may not actually be on top any more.

3. AP_TERM - 50 Version 4.0 and up
Request to terminate the current process

word 0 - 50
word 5 - Code identifying the reason of shuting down

This message is sent when the system requests that the application terminate. This occurs, for instance, when the user requests a resolution change. The response to this message should be to close windows, shut down, and terminate.
This message is sent when the system or other applications request the target process (application or accessory) to terminate (exit). This may due to a resolution change or simply the other application wants to take over the system.

Upon receiving this message and if the process decides to terminate itself, it should proceed with the normal termination sequences like closing all its windows, freeing the resources, free the menu etc.

However, process must inform the system if it chooses not to terminate or it has error that prevents it from terminating itself. Use shel_write mode 10 to inform the AES if such situation should arise.

From the caller point of view:

The caller should check the return code to see if the request is granted or denied.

If everything is OK, then caller should go back to its evnt_multi loop to wait for message SHUT_COMPLETED. This message will only be sent to the caller once. The message structure is defined as below:

word[0] = SHUT_COMPLETED (60)
word[3] = 1 - Shut down is succeeful, 0 - Shut down failed.

If there is an error i.e. word[3] is zero, then word[4] will contain the AES id of the process that has error and word[5] is the actual error code from that process. Otherwise, if the whole shutdown process is successful, the caller itself may choose to or not to exit the system depending on its initial purpose of calling this function.

From the receiver point of view:

Application willreceive AP_TERM message and accessory will receive the AC_CLOSE during the shutdown. If is is a complete shut down, the accessory will also receive AP_TERM message.

The message structure is defined as below:
word[0] = AP_TERM or AC_CLOSE
word[5] = Reason for shuting down.
For example: AP_TERM for just shuting down
AP_RESCHG for changing resolution

If receiver choose not to comply with the request, it should send a AP_TFAIL message to AES by using the shel_write(10,...) See also AP_TFAIL message.

4. AP_TFAIL - 51 Version 4.0 and up
Fail to terminate or close.

This message is sent when the receiver of AP_TERM or AC_CLOSE message decides not to close or terminate or it just has problem somewhere and would like to inform AES about the problem.

The message structure should look like:
word[0] = AP_TFAIL (51)
word[1] = Your error code

This message should be sent by using shel_write(10 ...).

5. AP_RESCHG - 57 Version 4.0 and up
Code identifying reason to terminate or close.

This message is used in conjunction with AP_TERM message.

6. SHUT_COMPLETED - 60 Version 4.0 and up

This message is sent to the initial caller of shuting down.
Please read also AP_TERM above.

7. RESCH_COMPLETED- 61 Version 4.0 and up
Message from AES to report the resolution change condition.

This message is sent to the caller who request the AES to do resolution change.

When a process requests to do a resolution change and the request is granted. It should wait on this message from the AES.

The message structure is defined as follow:
word[0] = RESCH_COMPLETED
word[3] = 1 for no error, 0 for error

If there is no error, the caller must exit the system in order to complete the resolution change process.

8. CH_EXIT - 80 Version 4.0 and up

word 0 - 80
word 3 - Child process's AES id
word 4 - Child's exit code

This message is sent back to the process when its child process is terminated.

Note: Process created by using the shel_write call will be chilren of the AES. AES remembers what process called shel_write and sends that process CH_EXIT to the caller when the process exits.

Discussion of wind_update

Wind_update is a semaphore that lets applications request control of the screen. The application will either be granted access to the screen or be put on hold until the current owner of the screen semaphore releases its control.

The purpose of wind_update is to let an application take control of the whole screen when it is writing data to the screen or its windows. This function is to ensure the application that after the wind_update(1) is successful, the status of the screen will remain the same until the wind_update(0) call. Other applications' window open, close, move, etc. calls will be suspended to prevent the state of the screen from changing while it is "locked." It is especially important to an application that the screen must not change when it is walking the rectangle list (using wind_get(WF_FIRSTXYWH,...) and wind_get(WF_NEXTXYWH,..)).

It is recommended that you call wind_update ONLY when necessary for walking the rectangle list and making VDI calls to write to the screen. Because other processes calling wind_update will block until you let go, it is not a good idea to hold onto the screen semaphore longer than necessary in a multitasking environment.

Traditionally, some programs have been written using a model like this:

```
/* lock the screen during initialization */
wind_update(TRUE);

...initialize...

/* main loop */

while (!done) {
      /* unlock the screen, wait for events */
      wind_update(FALSE);
      evnt = evnt_multi(...);

      /* lock the screen again and process events */
      wind_update(TRUE);

      if (evnt & MU_MESAG) {
            ...
      }
      if ...
}
```

This is not a good model for AES programming, because the screen is locked at all times, except when the program is blocked waiting for events in its evnt_multi call. A more correct model for AES programming is to leave the wind_update semaphore alone except at the very moment that you need it, which is during screen updates when you are walking the rectangle list.

Discussion of AES environment variables:

NAME	MEANING
ACCPATH	A comma-separated list of directories which will be searched for *.ACC at startup time. When an accessory is found in a given directory, that directory will be the accessory's default directory when it starts. The root directory of the boot device is always searched in addition to any directories appearing in ACCPATH.
PATH	A comma-separated list of directories which will be searched for programs when shel_write is called in mode 0, 1, or 3. See shel_write. In addition, shel_find and rsrc_load will look in all directories in this path when searching for files.
TOSEXT	A comma-separated list of extensions which are to be considered "TOS programs." See shel_write.
GEMEXT	A comma-separated list of extensions which are to be considered "GEM programs." See shel_write.
ACCEXT	A comma-separated list of extensions which are to be considered "accessories." See shel_write.

Discussion of AES.CNF

A file called AES.CNF may appear in the root directory of the boot device. It is a text file. Each line must begin in the first column. A line beginning with a number sign ("#") is a comment. A command in this file is separated from its argument with a single space.

The commands are:

setenv <NAME=VALUE>

This command places NAME=VALUE in the AES environment.

run <program>

This command causes AES to run the indicated program at boot time. The program is launched as a GEM program and is run concurrently, just as if you'd double-clicked it.

shell <program>

This command launches the indicated program, just like "run" does, but sets a flag inside AES which prevents it from launching the Desktop. You should use this command if you run a "desktop replacement" program and you don't want the Atari Desktop to appear.

AE_SREDRAW=0

The AES normally sends a full-screen redraw message when a GEM program starts up (calls appl_init). If this command is in AES.CNF that message will not be sent. This will soon be an environment variable instead.

AE_TREDRAW=0

The AES normally sends a full-screen redraw message when a GEM program finishes (calls appl_exit). If this command is in AES.CNF that message will not be sent. This will soon be an environment variable instead.

EXAMPLES:

```
setenv PATH=.,c:\bin,c:\gem
setenv TOSEXT=TOS,TTP
setenv GEMEXT=PRG,APP,GTP
setenv ACCEXT=ACC
setenv ACCPATH=c:\acc
run c:\mint\clock.prg
shell c:\mint\mw.prg
AE_TREDRAW=0
AE_SREDRAW=0
```

BUGS:

A program started with "run" or "shell" can't be given arguments.

The PATH doesn't get searched when a program is started with "run" or "shell."

COLOR ICON FORMAT

version: 2.1 Sept 8, 1992

The color icon feature is implemented in AES version 3.3 and up.

This document specifies the format for TOS color icons. The format features the ability to take advantage of all resolutions and to perform limited animation when an icon is selected. Below is a description of the actual data structure. The structure itself and how it is going to be used is explained in the description that follows. At the end of this document, a file format for a color icon resource file is outlined.

Data Structures:

```
typedef struct cicon_data {
      int      num_planes;    /* number of planes in the following data          */
      int      *col_data;     /* pointer to color bitmap in standard form        */
      int      *col_mask;     /* pointer to single plane mask of col_data        */
      int      *sel_data;     /* pointer to color bitmap of selected icon        */
      int      *sel_mask;     /* pointer to single plane mask of selected icon   */
                              /* next_res is a pointer to next icon for a        */
                              /* different resolution                            */
      struct cicon_data *next_res;
} CICON;
```

```
typedef struct cicon_blk {
      ICONBLK    monoblk;          /* default monochrome icon                    */
      CICON      *mainlist;        /* list of color icons for difrnt resolutns */
} CICONBLK;

#define G_CICON     33             /* object type number for AES                 */
```

Description:

The AES Object Library uses the CICONBLK structure to hold the data that defines color icons. The object type G_CICON points with its ob_spec pointer to a CICONBLK structure.

CICONBLK is a color icon block; a color icon block contains a monochrome icon block and a list of color icons. The list, mainlist, is a linked list of color icons that supports different resolutions. The monochrome icon block, monoblk, is the default icon displayed when mainlist does not contain an icon for the current resolution. Furthermore, the monochrome icon and all of the color icons in mainlist share the dimensions, placement, and all textual information contained in monoblk.

CICON is the structure that contains the color data. A CICON can contain pointers to two sets of icon data:

one for the color icon, col_data,
and one for the color icon in its selected state, sel_data.

In both cases, the data is an array of words, and is in device-independent format. The number of planes of data is determined by num_planes (Note: In mainlist, CICON's must have a unique num_planes.). Each CICON must have a valid pointer to data in col_data, but sel_data is optional. In other words, if sel_data is NULL, then when the icon is selected, the icon will be drawn darkened (i.e. dithered). Both col_data and sel_data pointers have masks: col_mask and sel_mask, respectively. Any other CICON's (with a different number of planes) are pointed to by next_res.

File Format:

The following is a description of the modifications to a GEM Application Resource File to include color icons. In the modifications are minor changes to the existing resource file header as well as the resource file itself. A new array of pointers for extensions to the resource file will be attached to the old file format, which will be followed by extension data. For color icons, the extension data consists of a table and color icon data.

The original specification of the Application Resource File consisted of a header which was followed by the structures described by the header. The first word of the header, rsh_vrsn, was always zero, but the new version of the header will now contain a value that has the third bit on (e.g. 0x0004).

The array is an extension array. Each long in the extension array is a specifically defined slot that contains information about extensions to the resource file. The first slot must contain the size of the actual file. The second slot is defined as the offset to color icon data structures. Other slots will be defined at a future date. If a slot contains a 0L, then that is the end of the array, and if a slot contains a -1L, then that slot's extension is not used. Hence, if a resource file contained color icons and no other extensions, the array would contain the file size in the first long, an offset in the second long, and a 0L in the third. Note that since the array contains long offsets, the resource file can now be larger than 64K. In addition, the array must start on a word boundary.

The offset found in the color icon slot will point to space reserved for a table of pointers. The table should have as many entries as there are CICONBLK's plus one. However, all the pointers should be initialized to 0L, except for the last entry which should have a -1L. After the last entry, the color icon data should follow. In general, the data should consist of one after another, preserving the structures within the resource file except for some pointer modifications (see below).

Within the actual Resource File, objects can now be of G_CICON type. Given an ob_type of G_CICON, the ob_spec will contain a zero-based value of which color icon to use. In other words, if an ob_spec contains a zero, the object will point to the first color icon. This scheme is similar to the monochrome icon format.

In summary, an Application Resource File that has color icons should consist of two sections: The first section should be a resource file as defined by prior AES documentation, except that objects can be of G_CICON type and that the first word is now an offset to the second section. The first section is immediately followed by the second which consists of space for an extension array, a table of pointers to color icons, and the color icon structures themselves. The following is a description of how the resource file, the table, and the color icon structures should look in the file (Note: Unless specified, all pointers may contain dummy values, and all offsets are relative to the beginning of the file.):

```
Resource File Header {
      WORD rsh_vrsn              /* should have bit 2 on                          */
      WORD rest_of_header[17]    /* rest of the rsc file header, unchanged */
}



Resource File {
/* standard format, except objects may be of ob_type G_CICON (see above) */
}



Table of Extensions {
      LONG filesize           /* size of the file                                */
      LONG color_ic           /* slot for color icons, containing an offset to
                               * Table CICONBLK's                                */
      LONG dummy[?]           /* more extension slots, to be defined             */
      LONG end_extensions /* always 0L                                           */
}



Table of CICONBLK's with (number of CICONBLK's) entries {
      0L
      0L
      ...         (last entry of color icons)
      -1L         (end of table marker, this entry will never be converted to
                  actual memory pointer)
}
```

```
for ( number of CICONBLK's ) {
      ICONBLK - (monochrome icon, same as the AES definition) {
        +0  LONG     - monochrome mask pointer
         4  LONG     - monochrome data pointer
         8  LONG     - text, 0L if no text
        12  WORD[5]  - color info, character, placement vectors
        22  WORD     - width in pixels
        24  WORD     - height in pixels
        26  WORD[4]  - dimension and placement of text box
            }

      34  LONG     -  number of CICON's of different resolutions

    +38   WORD[n]  - monochrome bitmap data, where n = # of words in mono icon
          WORD[n]  - monochrome mask data, where n = # of words in mono icon
          BYTE[12]- text string (max of 12 characters)

          for (number of different resolution CICON's) {
            +0  WORD     - number of planes
            +2  LONG     - color data pointer
             6  LONG     - color mask pointer
            10  LONG     - select data pointer, if 0L then no extra data or mask
                          follows
            14  LONG     - select mask pointer
            18  LONG     - next_res, 1L indicates more icons to follow

            22  WORD[n]  - color data,
                          where n = (# of words in mono icon) * num_planes
                WORD[n]  - mask data, where n = # of words in mono icon

                if (select data pointer ) {
                    WORD[n]  - select data,
                               where n = (# of words in mono icon) * num_planes
                    WORD[n]  - select mask, where n = # of words in mono icon
                    }
          }          /* end different resolutions */
}         /* end number of CICONBLK's */
```

Addendum:

Restriction on Color Icon Data

Due to a bug found in the VDI, the seventh word in the color icon data cannot contain 0x0001. The VDI looks at the wrong location to see whether or not it should skip a transform form on the icon data. The end result is that icons that have the seventh word of data set to 0x0001 will be blitted in the wrong data format. For those who are writing icon editors, a check on the data is recommended to prevent this strange behavior.

Since this is a rather obscure occurence, icon designers should not need to worry about this problem unless a particular icon does not display correctly (i.e. the data is scrambled). If the icon is incorrectly displayed, change the image so that the first sixteen pixels on the fourth line contain different data, until the image becomes unscrambled.

Rsh_vrsn

The first word in the resource file header, rsh_vrsn, no longer contains the offset to the table of extensions. The third bit must be set to 1 (i.e. 0100).

Table of Extensions

Please note, the table of extensions must begin on a word boundary.

Reuse of Icons

If no appropriate color icon exists for the current video mode, AES will now take four-plane icons and reuse the data in eight-plane and true color mode. In addition, eight-plane icon data will be converted to true-color mode. Therefore, a deskicon.rsc containing only four-plane icons is sufficient for all modes above four planes.

Color Icon Table

The color icon table is now has a -1 terminator, so that it is initialized with 0L for all icon slots along with the extra -1L as a terminator.

IDT

International Day and Time Cookie

9/10/92

Beginning from AES 3.3 and up, the desktop will use the _IDT cookie to determine how to set up the date and time format. The _IDT should look like the following.

_IDT LONG WORD:

	HIGH WORD	LOW WORD		
Bit Number:	31 - 16	15-12	11-8	7-0 bit
Meaning:	Reserved	Time	Date	Date Seperator
Value:		0 12 hour	0 MM-DD-YY	ASCII value
		1 24 hour	1 DD-MM-YY	(A zero value equivalent
			2 YY-MM-DD	to '/')
			3 YY-DD-MM	

The current default values are:

Country	Time	Date	Seperator
USA	0	0	/
Germany	1	1	.
French	1	1	/
Spain	1	1	/
Italy	1	1	/
Sweden,Norway			
Finland	1	2	-